Fibonacci

**def** fib(n):  
 **if** (n==0):  
 **return** 0  
 **if** (n==1 **or** n==2):  
 **return** 1  
 **else**:  
 **return** fib(n-1)+fib(n-2)  
n=int(input(**'Enter an Integer:'**))  
print(**'The Fibonacci sequence of given integer is:'**,end=**""**)  
**for** i **in** range(0,n):  
 print(fib(i),end=**","**)

doubly

**class** node:  
 **def** \_\_init\_\_(self,data):  
 self.prev=**None** self.next=**None** self.data=data  
**class** dlist:  
 **def** \_\_init\_\_(self):  
 self.head=**None  
 def** printlist(self):  
 print(**"\n Trversal in forward direction"**)  
 node=self.head  
 **while**(node **is not None**):  
 print(node.data)  
 last=node  
 node=node.next  
 print(**"\n Traversal is reverse direction"**)  
 **while** last:  
 print(last.data)  
 last=last.prev  
node1=node(**'Mon'**)  
node2=node(**'Tues'**)  
node3=node(**'Wed'**)  
dlist=dlist()  
dlist.head=node1  
dlist.head.prev=**None**dlist.head.next=node2  
node2.prev=dlist.head  
node2.next=node3  
node3.prev=node2  
print(**'The doubly linked list is:'**)  
print(**'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'**)  
dlist.printlist()

fact

**def** factorial(n):  
 **if**(n==0 **or** n==1):  
 **return** 1  
 **else**:  
 **return** n\*factorial(n-1)  
n=int(input(**'Enter an Integer:'**))  
result=factorial(n)  
print(**'The factorial of given integer is:'**,result)

queue

queue=[]  
**def** enqueue():  
 item=input(**"Enter the item:"**)  
 queue.append(item)  
 print(item,**'is inserted to queue'**)  
**def** dequeue():  
 **if not** queue:  
 print(**'Underflow (empty Queue)'**)  
 **else**:  
 item=queue.pop(0)  
 print(**'Deleted item='**,item)  
**def** display():  
 print(queue)  
**while True**:  
 print(**"Select the operation 1.Insert 2.Delete 3. Display 4.Quit \n"**)  
 choice=int(input())  
 **if** choice==1:  
 enqueue()  
 **elif** choice==2:  
 dequeue()  
 **elif** choice==3:  
 display()  
 **elif** choice==4:  
 **break  
 else**:  
 print(**'Enter Correct operation'**)

queue

queue=[]  
**def** enqueue():  
 item=input(**"Enter the item:"**)  
 queue.append(item)  
 print(item,**'is inserted to queue'**)  
**def** dequeue():  
 **if not** queue:  
 print(**'Underflow (empty Queue)'**)  
 **else**:  
 item=queue.pop(0)  
 print(**'Deleted item='**,item)  
**def** display():  
 print(queue)  
**while True**:  
 print(**"Select the operation 1.Insert 2.Delete 3. Display 4.Quit \n"**)  
 choice=int(input())  
 **if** choice==1:  
 enqueue()  
 **elif** choice==2:  
 dequeue()  
 **elif** choice==3:  
 display()  
 **elif** choice==4:  
 **break  
 else**:  
 print(**'Enter Correct operation'**)

stack

stack=[]  
**def** push():  
 **if** len(stack)==n:  
 print(**"Overview (stack is full)"**)  
 **else**:  
 item=int(input(**"Enter item to push:"**))  
 stack.append(item)  
 print(stack)  
**def** pop():  
 **if not** stack:  
 print(**"Underflow (empty stack)"**)  
 **else**:  
 item=stack.pop()  
 print(**"Popped element="**,item)  
 print(stack)  
n=int(input(**"Enter stack limit:"**))  
**while True**:  
 print(**'Select the operation 1.Push 2.Pop 3.Quit'**)  
 choice=int(input())  
 **if** choice==1:  
 push()  
 **elif** choice==2:  
 pop()  
 **elif** choice==3:  
 **break  
 else**:  
 print(**'Enter Correct Operation'**)

circular

**class** node:  
 **def** \_\_init\_\_(self,data):  
 self.data=data  
 self.next=**None  
class** circularlinkedlist:  
 **def** \_\_init\_\_(self):  
 self.head=**None  
 def** printlist(self):  
 temp=self.head  
 **if** self.head **is not None**:  
 **while**(**True**):  
 print(temp.data,end=**"->"**)  
 temp=temp.next  
 **if**(temp==self.head):  
 **break**clist=circularlinkedlist()  
node1=node(**'Mon'**)  
node2=node(**'Tues'**)  
node3=node(**'Wed'**)  
node4=node(**'Thur'**)  
clist.head=node1  
clist.head.next=node2  
node2.next=node3  
node3.next=node4  
node4.next=clist.head  
print(**'The circular linked list is:'**)  
print(**'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'**)  
clist.printlist()

Tree

**class** Node:  
 **def** \_\_init\_\_(self,data):  
 self.data=data  
 self.left=**None** self.right=**None  
class** BinarySearchTree:  
 **def** \_\_init\_\_(self):  
 self.root=**None  
 def** insert(self,value):  
 newNode=Node(value)  
 **if** self.root **is None**:  
 self.root=newNode  
 **else**:  
 curNode=self.root  
 **while** curNode **is not None**:  
 **if** value<curNode.data:  
 **if** curNode.left **is None**:  
 curNode.left=newNode  
 **break  
 else**:  
 curNode=curNode.left  
 **else**:  
 **if** curNode.right **is None**:  
 curNode.right=newNode  
 **break  
 else**:  
 curNode=curNode.right  
 **def** preorder(self,rt):  
 print(rt.data,end=**" "**)  
 **if** rt.left **is not None**:  
 self.preorder(rt.left)  
 **if** rt.right **is not None**:  
 self.preorder(rt.right)  
 **def** postorder(self,rt):  
 **if** rt.left **is not None**:  
 self.postorder(rt.left)  
 **if** rt.right **is not None**:  
 self.postorder(rt.right)  
 print(rt.data,end=**" "**)  
 **def** inorder(self,rt):  
 **if** rt.left **is not None**:  
 self.inorder(rt.left)  
 print(rt.data,end=**" "**)  
 **if** rt.right **is not None**:  
 self.inorder(rt.right)  
bst=BinarySearchTree()  
ls=[25,10,35,20,65,45,24]  
**for** i **in** ls:  
 bst.insert(i)  
print(**"\nPre-order"**)  
bst.preorder(bst.root)  
print(**"\nPost-order"**)  
bst.postorder(bst.root)  
print(**"\nIn-order"**)  
bst.inorder(bst.root)